Course de voiture en réseau DTN

27 mars 2011

Responsable : M. Serge Chaumette
Auteurs : Bruno Voisin,
Papa Samba Diop,
Eddy Chen,
Vidal Watat

Table des matières

I Introduction 2
II Rappel du sujet 3
III Technologies utilisées 4
III.1 Choix technologiques 4
III.2 Utilisations 4
III.2.1 Installation de l'environnement de développement Android 4
Installation du SDK 4
Installation du ADT Plugin pour Eclipse 5
III.2.2 Utilisation de SVN 5
IV Description du travail à réaliser 7
V Aspects techniques 9
V.1 Le jeu 9
V.2 L'interface graphique 10
V.3 Le réseau DTN 11
V.3.1 Connexion 11
V.3.2 Transfert des données 12
V.3.3 Synchronisation 13
V.4 Le wifi Ad-hoc 14
V.5 Architecture 15
VI Résultats obtenus 17
VII Travail restant à faire et extensions possibles 18
VIII Manuel utilisateur 19
IX Conclusion 24

Références **25**

I Introduction

L'avènement des terminaux mobiles exigent d'expérimenter de nouvelles façons d'implémenter les communications inter-réseaux entre ces derniers avec une plateforme de développement évoluée. La communication et la mobilité de ces terminaux entrainent une contrainte de connexion qui peut entraver le fonctionnement des applications installées dans ces terminaux.

Dans ce contexte, il devient nécessaire d'implémenter un concept qui permettra de répondre aux pertes de connectivité qui pourront éventuellement se produire.

Le but de ce projet consiste donc à mettre en place un jeu de course de voiture sur la plateforme Android avec un réseau tolérant au délai (DTN). L'objectif est de donner l'impression à l'utilisateur qu'il joue en temps réel avec son adversaire alors que les données sont transmises sur un DTN.

Le travail effectué dans ce document consiste dans un premier temps à dresser la description du sujet et à justifier nos choix techniques. Dans un second temps, nous décrirons le travail qui a été réalisé du point de vue technique et les résultats qui ont été obtenus. Pour une présentation de l'application et de ses fonctionnalités, un manuel d'utilisation s'avère nécessaire.

II Rappel du sujet

Les réseaux DTN sont des réseaux dans lesquels les messages que l'on cherche à échanger le sont avec des délais parfois longs et peuvent même être perdus. Dans ce contexte on souhaite réaliser une application de course de voitures multi-utilisateurs avec les caractéristiques suivantes :

- Le circuit comporte des obstacles qu'il faut éviter. En cas de choc avec un obstacle la voiture concernée est temporairement arrêtée ou ralentie.
- Chaque utilisateur dispose d'une tablette Android lui permettant de visualiser la course et de contrôler sa voiture (par inclinaison de la tablette). Des messages sont échangés entre les tablettes pour permettre de connaître la position de la voiture adverse.
- La connexion ou des messages échangés entre les tablettes peuvent être perdus mais ceci ne doit pas être visible pour les utilisateurs (la machine prend le relai de l'utilisateur non accessible et la course se poursuit de manière transparente).
- Lorsqu'une connexion défaillante est rétablie (ou remplacée par une connexion avec un autre utilisateur que l'utilisateur original) l'application vue des utilisateurs doit là aussi se poursuivre de manière transparente.

Le travail demandé consiste à concevoir et mettre en œuvre cette application.

III Technologies utilisées

Cette partie est constituée des choix technologiques et explique comment les mettre en place.

III.1 Choix technologiques

Comme indiqé dans le sujet, nous allons développer l'application sous Android. Nous avons aussi choisi de programmer en Java parce que la documentation est proposée pour la plupart dans ce langage. Pour pouvoir utiliser Android il nous sera nécessaire d'installer le SDK d'Android.

Nous avons choisi l'environnement de développement Eclipse parce qu'il dispose d'un plugin ADT (Android Development Tools) facilitant le développemnet d'application Android.

De plus, pour un travail collaboratif, nous avons choisi d'installer SVN sous Eclipse.

Pour les communications réseaux, nous utilisons le protocole UDP, parce que notre application n'a pas besoin de vérifier la communication de bout en bout. De plus le protocole UDP est plus léger et plus simple.

III.2 Utilisations

Dans cette partie, nous décrivons comment nous avons mis en œuvre nos choix technologiques.

III.2.1 Installation de l'environnement de développement Android

L'installation de l'environnement de développement Android sous Eclipse est bien documentée sur `http://developer.android.com/` donc facile à mettre en place.

Installation du SDK

1. Télécharger le SDK d'Android pour Linux sur `http://developer.android.com/sdk/index.html`
2. Décompresser et mettre à jour la variable d'environnement dans .bashrc
3. Mettre à jour le SDK : android update sdk

Installation du ADT Plugin pour Eclipse

Maintenant il faudra installer le plugin sous Eclipse : Help > Intall new software

1. dans la boite de dialogue cliquez sur le bouton Add
2. dans la boite de dialogue Add site, mettre un nom dans le champ name, puis dans le champ location mettre l'url : `https://dl-ssl.google.com/android/eclipse/`
3. dans la boite de dialogue Aviable software, cocher la case Developper Tools et cliquer sur Next
4. dans la fenêtre Install Details, les plugins doivent être présents. Ensuite cliquer sur Next et enfin Finish
   Il faut ensuite redémarrer Eclipse. Ensuite il faut faire en sorte que Eclipse utilise le SDK :
5. Aller dans Window > Preferences
6. Choisir Android à gauche
7. Dans le champs SDK Location, cliquez sur Browse... et sélectionnez le répertoire dans lequel vous avez installé le SDK Android.
8. Cliquez sur Apply, puis OK.

III.2.2 Utilisation de SVN

Nous avons créé un dépôt SVN pour un travail collaboratif plus performant. L'URL d'accès au dépôt SVN est :
svn+ssh ://jaguar.emi.u-bordeaux1.fr/net/travail/uti/brvoisin/hidden/svncoursedtn

1. Créez un nouveau projet « Projet depuis SVN » ou « Checkout Projects from SVN »
2. Entrez l'URL du dépôt dans les informations de l'emplacement du référentiel et cliquez sur « Suivant ».
3. Une authentification sur jaguar.emi.u-bordeaux1 est nécessaire, entrez votre login et votre mot de passe. (Il peut être demandé plusieurs fois.)
4. Dans la sélection de la ressource « Head Revision » est déjà coché, cliquez sur « Finish »
5. Choisissez de rechercher des projets Eclipse dans les sous-dossiers de la ressource et cliquez sur « Finish ».
6. Enfin, cochez « Extraire comme des projets dans l'espace de travail » et cliquez sur "Finish".
7. Un mot de passe est demandé pour sécuriser le stockage.

Les actions relatives à SVN (update, commit, etc.) sont disponibles dans le menu « Team » après avoir fait un clique droit sur le projet ou un fichier du projet.

IV Description du travail à réaliser

Comme précisé plus haut, le travail consiste à réaliser un jeu de course de voiture en réseau sur des tablettes Android. Pour cela nous devons implémenter une interface pour le jeu avec toutes les fonctionnalités demandées par le sujet.

Au démarrage de l'application, nous devons avoir une interface qui permet au joueur d'entrer son pseudonyme et d'appuyer sur le bouton « **Jouer** » pour démarrer le jeu. Une fois sur le jeu, la vue de la tablette sera divisée en deux parties. La partie gauche représentant le joueur local et la partie droite représentant le joueur distant ou virtuel. De plus, le jeu sera constitué d'un circuit générant des obstacles en forme de rocher à position aléatoire.

Si la voiture heurte un obstacle, elle change de couleur et le joueur perd des points. Le calcul du score ne doit pas dépendre du temps de jeu, mais de l'habilité du joueur. Cela permet de rendre la partie équitable entre un joueur qui vient de se connecter et un joueur qui a duré dans le jeu. Nous devons aussi mettre en place le gyroscope. En d'autres termes, le joueur local doit pouvoir modifier la position de la voiture par le mouvement d'inclinaison de la tablette.

La deuxième grande partie du travail consistera à mettre en place le réseau dans notre application. Il faudra d'abord implémenter la gestion de réseau Wifi ad hoc qui n'est pas faite sous Android. Une fois que la connexion Wifi ad hoc établie, le joueur local qui joue avec le joueur virtuel envoie aussi des broadcasts UDP afin de savoir s'il y a un autre joueur qui s'est connecté au jeu. Ainsi, ce joueur distant devra se synchroniser de manière transparente et ainsi remplacer le joueur virtuel. Les deux voitures doivent aussi se communiquer périodiquement leur position et leur score. Nous devons aussi gérer la déconnexion et la reconnexion des joueurs. Autrement dit, un joueur doit pouvoir se déconnecter via un bouton de l'interface et se reconnecter de façon transparente à l'utilisateur. Il semble donc nécessaire d'élaborer un algorithme de synchronisation afin de pouvoir assimiler les points importants de cette partie. Une fois la connexion établie, la synchronisation commence immédiatement. Il y a trois points traités lors de cette synchronisation :

- **La position** : nous allons synchroniser la position uniquement sur l'axe des x (gauche-droite). La synchronisation en y peut être traitée avec les obstacles. On remplace immédiatement le joueur virtuel par le joueur distant mais on ne lui affecte pas tout de suite la position reçue. On contrôle sa position jusqu'à ce qu'elle corresponde à celle reçue. Si la position reçue est à gauche par rapport à la position affichée, on se déplace sur la gauche ; si elle est à droite,

on se déplace à droite. Ceci jusqu'à faire correspondre les deux positions. La synchronisation est finie, on peut continuer normalement, c'est-à-dire affecter la position reçue à celle du joueur distant.

- **Les obstacles** : cette partie est la plus difficile. Les obstacles du joueur distant sont reçu dès le début de la synchronisation. Pour que la mise à jour des obstacles ne soit pas visible à l'utilisateur, on effectuera les changement uniquement dans la partie non affichée du circuit. La partie nouvellement affichée du circuit contient donc les obstacles du joueur distant. La partie sortant de l'écran est mise à jour jusqu'à ce que le circuit corresponde totalement au distant.
- **Le score** : pour synchroniser le score nous allons placer un obstacle spécial inévitable à la limite de la partie visible de l'écran devant la voiture. Lorsque la voiture entre en collision avec cet obstacle le score réel du joueur distant sera affecté au joueur distant. Des obstacles spéciaux seront affichés de façon aléatoire pour ne pas se rendre compte d'une synchronisation.

V Aspects techniques

Nous allons détailler les aspects techniques en trois parties. La première concerne l'implémentation de la logique du jeu, la deuxième, l'interface graphique et la troisième, le réseau.

V.1 Le jeu

Le jeu est implémenté par la classe GameImpl dont l'interface est Game. Il peut contenir plusieurs niveau (classe Level). Les méthodes start() et stop() permettent respectivement de démarrer et arrêter le jeu. Le jeu démarré avance pas à pas grâce à la méthode onStep() appelée périodiquement avec un certain délai (30 millisecondes) défini dans la classe Config.

Un niveau comporte deux circuits (interface Circuit). L'un est celui du joueur local et l'autre de l'adversaire. Chaque circuit possède son ensemble d'obstacles (classe Obstacle) et son joueur (interface Player). Un circuit est un rectangle où les obstacles et le joueur ont une position. L'origine du repère est en bas à gauche du circuit et le sens de la course est de bas en haut. Le circuit est une route droite mais circulaire puisqu'arrivée au bout la voiture repart au début et un tour a été effectué.

Le nombre de tours d'une course est défini dans la classe Config et si sa valeur est zéro il n'y a pas de limite, la course est infinie. Lorsque la voiture a fini ses tours l'exception EndOfGameException est levée pour arrêter le jeu si les deux circuits sont terminés.

La méthode onStep() d'un circuit est appelée par la méthode du même nom de GameImpl. Elle appelle à son tour onStep() du joueur pour qu'il avance d'un pas. Le pas effectué sur l'axe des y est la vitesse en y de la voiture et elle est constante (définit dans la classe Config). Ensuite, le circuit vérifie les collisions entre la voiture et les obstacles. Elle vérifie également si la voiture dépasse des obstacles sans les toucher et calcule le score.

Le score correspond à la moyenne de l'habilité du joueur sur des intervalles de temps bien déterminé. L'habilité étant le rapport entre le nombre d'obstacles évités et le nombre total d'obstacles dépassés (évités et heurtés).

V.2 L'interface graphique

L'interface graphique est séparée en deux parties, chacune décrit par un fichier XML qui définit les éléments graphiques.

La première (res/layout/main.xml) est un simple menu permettant de saisir son pseudonyme et de lancer le jeu via un bouton « **Jouer** ».

La deuxième (res/layout/game.xml) est la vue du jeu. Elle est constituée de deux vues : à gauche le circuit du joueur local et à droite celui du joueur distant ou virtuel. Ces deux vues sont des CricuitViews.

Tout l'affichage d'un circuit (obstacles et voiture compris) se fait dans la classe CircuitView. La position sur l'axe des y de la voiture reste constante à l'écran et l'affichage du reste se fait par rapport à la position de la voiture dans le circuit. Les obstacles vont donc descendre à l'écran mais leur position réelle est fixe. Le pseudonyme et le score sont affichés en haut de l'écran.

La difficulté (surmontable) est le changement de repère entre les coordonnés du jeu et celles de la vue (écran) qui de plus bouge sur l'axe des y. La direction de l'axe des y est inversée dans le repère du jeu par rapport à celui de la vue.

L'interface est aussi consituée de boutons. Dans la partie gauche c'est à dire celle du LocalPlayer, un bouton avec l'icône du wifi est présent permettant d'activer ou de désactiver le wifi. Dans la partie droite, trois boutons permettent de basculer sur les trois modes synchronisation étayés dans la partie suivante (cf section V.3.3). Un bouton est présent sur la voiture elle même permettant d'afficher le pseudonyme de l'adversaire. Ces boutons ont été implémentés à l'aide de la méthode onTouchEvent(MotionEvent event) héritant de la classe View et qui détecte le toucher de l'écran sur une zone bien délimitée.

Le controle du joueur local est géré par le gyroscorpe. En effet l'API d'Android propose une classe SensorManager, qui permet de détecter les changements d'orientation subis par la tablette. Pour cela nous avons utilisé la classe abstraite getSystemService(). Elle permet l'interaction entre le matériel et le service. Ici, le service associé à la classe abstraite est de type SENSOR_SERVICE habilité à détecter les changements de direction. Pour connaître les changements de direction, il faut associer un listener au sensor en implémentant l'interface SensorEventListener. Nous avons utilisé l'angle en y de la méthode onSensorChanged() de cette interface afin de récupérer les variations de direction.

V.3 Le réseau DTN

La partie réseau du jeu est implémentée dans le paquetage ubx1.project.network.

La classe NetworkManager est la classe principale qui s'occupe d'établir et de gérer la connexion. Elle étend la classe Runnable qui permet de lancer un thread pour la réception des paquets. Les méthodes getLocalIpAddress() et getBroadcastAddress() permettent de renvoyer respectivement l'adresse IP de la machine et l'adresse de broadcast du réseau. Le calcul de l'adresse de broadcast est nécessaire car c'est elle qui permettra d'envoyer un message de découverte à toutes les tablettes connectées sur le réseau. C'est d'ailleurs la raison pour laquelle elle a été appelée dans le constructeur de NetworkManager et dans la méthode sendBroadcast() qui sera renvoyée à chaque fois que l'application cherche à se connecter à un autre joueur.

V.3.1 Connexion

L'établissement de la connexion se fait de la même façon qu'en TCP grâce aux méthodes sendSyn(), sendSynAck() et sendAck(). Ces méthodes sont appelées dans la méthode connexion() pour envoyer les paquets relatifs à la connexion. Une socket de reception est ouverte avec un « timeout ». Si elle reçoit un SYN, un SAC (Syn Ack) est envoyé et si elle reçoit un SAC, un ACK est envoyé pour que la connexion soit définitivement établie. Si un des paquets qui doivent être reçu pour la connexion n'est pas reçu après un certain « timeout », alors la connexion est abandoné et on recommence.

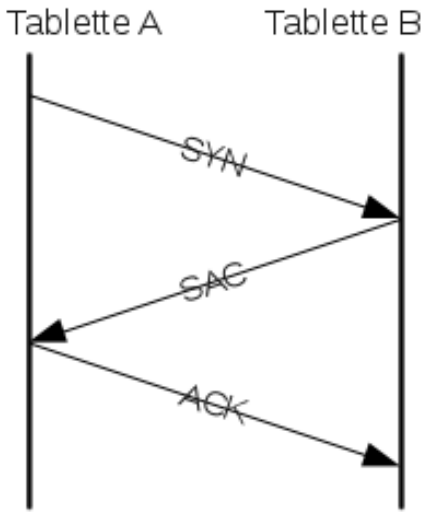

FIGURE 1 – Diagramme de la connexion

V.3.2 Transfert des données

La classe ByteArrayUtil permet de convertir les données qui sont des float ou des int en octet et vice versa. Cette classe sera utile à la classe Data et SynchData qui se chargent de transformer des informations en octets afin de pouvoir les envoyer sur le réseau.

Data contient les données à envoyer ou à recevoir périodiquement, c'est-à-dire la position et le score du joueur. Elle est utilisée par la méthode receiveData() de la classe NetworkManager pour convertir les octets reçus représentant la position et le score du joueur distant. Elle est également utilisée dans la méthode send(Player) pour pouvoir construire le paquet contenant le score et la position du joueur passé en paramètre.

FIGURE 2 – Diagramme des datagrammes Data

La classe SynchData contient les coordonnées des obstacles et le pseudonyme du joueur distant et est utilisée pour la synchronisation (cf section V.3.3). L'envoi et

la réception de ces données sont effectués dans la méthode startSynchronizing() de NetworkManager.

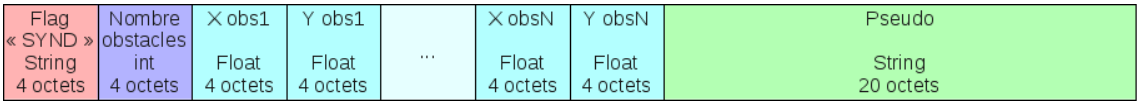

FIGURE 3 – Diagramme des datagrammes SynchData

V.3.3 Synchronisation

Une fois la connexion réseau établie, il faut implémenter la synchronisation de façon conforme afin de rendre transparent à l'utilisateur, le changement de joueur adverse c'est-à-dire le passage du joueur virtuel au distant. L'autre sens est trivial, le joueur virtuel garde les mêmes données que le distant qui se déconnecte. Dans ce contexte, il faudra synchroniser la position et le score du joueur mais aussi les obstacles du circuit. Le principe est d'affecter les données distantes au circuit adverse local de façon plus ou moins transparente.

Nous avons implémenté trois niveaux de synchronisation, de la plus basique à la plus transparente aux yeux de l'utilisateur.

La **synchronisation instantanée** affecte les données du joueur distant instantanément au joueur adverse lors de sa connexion. Le changement est visible puisque la position du joueur, son score et les obstacles sautent immédiatement pour correspondre au joueur distant.

La **synchronisation instantanée cachée** repose sur le même principe mais un brouillard apparaît pour masquer la synchronisation instantanée. Le joueur virtuel reste présent jusqu'à ce que le brouillard devienne opaque et à ce moment tout les changements sont effectués. L'apparition de brouillard se fait également de façon aléatoire pour ne pas savoir qu'il y a synchronisation.

La **synchronisation transparente** doit avoir le comportement décrit dans la partie « Description du travail à réaliser ».

Dès la connexion le joueur distant prend la place du virtuel mais en mode de synchronisation. C'est à dire qu'il part de la même position que le virtuel et effectue un mouvement vers la position reçue via le réseau. Ce mouvement ne concerne

que l'axe des x. Pour l'axe des y lisez le paragraphe plus bas concernant les obstacles. Une fois les deux positions x égales, le joueur distant n'est plus en mode de synchronisation et se voit affecté directement la position x reçue périodiquement.

Pour la synchronisation des obstacles, la difficulté vient du fait que les joueurs n'ont pas la même position dans le circuit. On calcule donc le saut, qui est la distance entre la position y de la voiture représentant le joueur distant localement et la position y réel de la voiture distante. Il faut donc insérer les obstacles en ajoutant ce saut à leur position y pour avoir le même affichage sur les deux tablettes. La position y de la voiture reçu via le réseau doit aussi prendre en compte ce saut. Ainsi il y a un décalage entre les deux circuits mais l'affichage est le même. On évite ainsi que la voiture fasse un saut dans le circuit.

Le score distant reçu est ignoré tant que la voiture n'a pas rencontré l'obstacle spécial ajouté en haut de l'écran au début de la synchronisation. Des obstacles spéciaux sont aussi ajoutés selon une certaine probabilité et représente un certain nombre d'obstacle évités ou non. La rencontre d'un tel obstacle simulera une collision ou une esquive d'obstacles ordinaires pour changer la valeur du score.

V.4 Le wifi Ad-hoc

Le Wifi en mode ad-hoc n'a pas besoin d'un point d'accès comme le mode en infrastructure. La connexion est directe entre plusieurs tablettes. Ce mode est donc idéal pour jouer à un jeu en réseau local sans fil n'importe où. Mais les développeurs d'Android n'ont pas implémenté la possibilité de créer ou de se connecter à un réseau Wifi ad-hoc.

Nous avons fait des recherches pour combler ce manque. Le programme wpa_supplicant est responsable de la gestion du Wifi. Nous avons trouvé une version modifiée de ce programme qui permet de voir et de se connecter à un réseau Wifi ad-hoc. Nous n'avons malheureusement pas le code source. Après avoir obtenu les droits nécessaire grâce à l'application Universal Androot et Root Explorer pour Android, nous avons pu remplacer le binaire wpa_supplicant. Le problème est que l'interface utilisateur d'Android ne permet pas de créer un réseau Wifi. Nous utilisons donc un PC portable pour créer un réseau Wifi ad-hoc afin de pouvoir connecter les tablettes.

Une autre piste permettrait de ne plus utiliser un PC portable pour initier la connexion Wifi. Une application open source se nommant Barnacle Wifi Tether permet de créer un réseau Wifi ad-hoc. Le problème est que toutes les applications « Wifi Tether » fonctionnent uniquement sur Android 2.2 et supérieur. Nos

tablettes sont à la version 2.1. Nous avons pu récupérer les sources d'Android et les compiler mais nous ne savons pas comment l'installer sur la tablette.

V.5 Architecture

Nos classes Java sont regroupées en deux paquetages game et network. Le premier contient toutes les classes relatives à l'implémentation du jeu et le deuxième celles implémentant la partie réseau de l'application.

Nos avons utilisé des interfaces (Game, Level, Circuit et Player) et une factory pour permettre le changement facile de l'implémentation.

La figure 4 page 16 est le diagramme de classe. En vert sont représentées des classes de l'API Android que nous utilisons.

Quelques remarques peuvent être faites sur cette architecture. L'interface Level a été conçue pour pouvoir enchaîner plusieurs niveaux. Elle n'est en fait pas utile dans notre jeu puisque nous avons qu'un seul niveau. De plus les deux paquetages sont en doubles dépendances. Il est préférable de n'avoir qu'une dépendance. Enfin, la classe NetworkManager à grossi progressivement au fil du développement. Pour un souci de clarté donc de maintenance il est nécessaire de faire du refactoring pour séparer la gestion des sockets et la synchronisation dans deux classes distinctes.

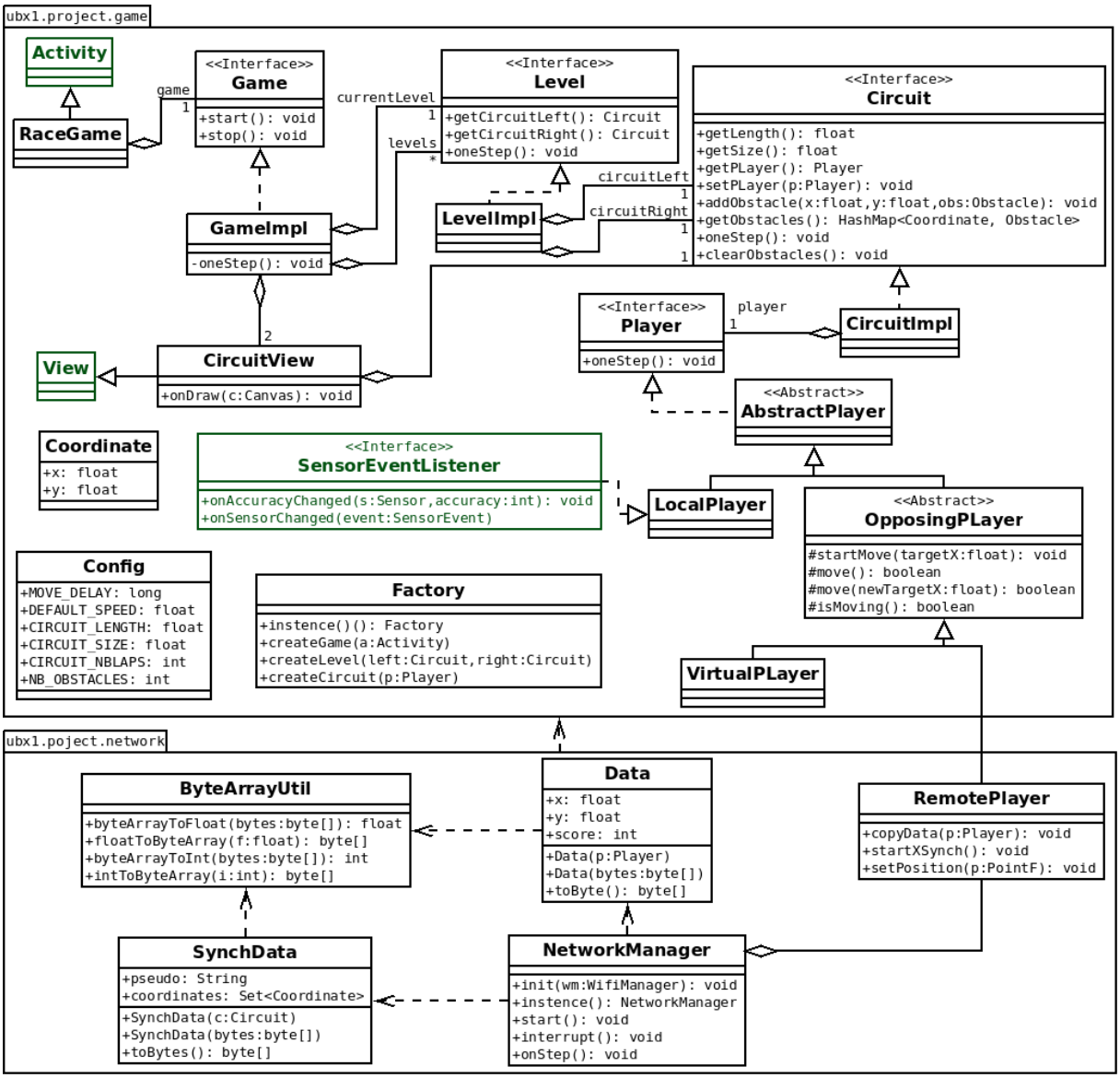

FIGURE 4 – Diagramme de classes

VI Résultats obtenus

Le jeu est fonctionnel et sans bugs connus. L'interface graphique est conforme aux besoins. Le contrôle de la voiture via l'inclinaison de la tablette est opérationnelle. Le déplacement du joueur virtuel est basique mais fluide. Ses mouvements sont aléatoires donc elle n'anticipe pas d'esquive d'obstacle. L'aspect DTN, c'est à dire les connexions et déconnexions, fonctionne correctement. Les trois modes de synchronisation sont implémentés et disponibles.

Nous avons effectué des tests avec trois tablettes, la connexion entre les deux tablettes n'est pas perturbée et le troisième joueur joue contre le joueur virtuel. Ce dernier est en attente d'une déconnexion. Lorsque l'un des deux joueurs se déconnecte le troisième joueur prend sa place de façon transparente.

Nous avons aussi testé la déconnexion par éloignement. Nous avons constaté que plus le joueur s'éloigne plus des paquets sont perdus et plus il y a des déconnexions et reconnexions répétitives. Malgrè ces déconnexions, l'application reste stable.

VII Travail restant à faire et extensions possibles

Les objectifs dressés dans le cahier des charges ont été atteints. Etant donné que notre application se joue dans un réseau Ad-hoc, il serait judicieux de pouvoir créer un réseau Ad-hoc directement à partir des tablettes. Mais il s'avère que les tablettes que l'on nous a fourni ne permettent pas de créer un réseau. En effet la version 2.1 de Android est celle qui est installée dans nos tablettes. Cette version ne permet pas d'installer des applications Wifi Tether qui sont sous la version 2.2. Une autre solution serait de recupérer les sources d'Android et de les compiler (ce que nous avons fait). Mais le manque de documentation sur ce sujet ne nous permet pas de savoir comment l'installer.

Par ailleurs, d'éventuelles extensions peuvent être envisagées afin d'améliorer le jeu.

- On peut ajouter la possibilité de faire une pause pendant une course.
- Il est possible d'améliorer la façon de quitter la course. Le score et le temps de jeu pourrait être affichés et sauvegardés.
- La vitesse de la voiture pourrait dépendre du score ou de l'inclinaison de la tablette en avant.
- Un système de combos pourrait faire varier l'importance d'une esquive sur le score. Par exemple, plus on évite d'obstacle à la suite plus le score augmente vite.
- On pourrait aussi ajouter plusieurs types d'obstacles et des bonus.

VIII Manuel utilisateur

Course DTN est un jeu à deux joueurs sous tablette Android représenté par une icône « Course DTN » dans le dossier application.

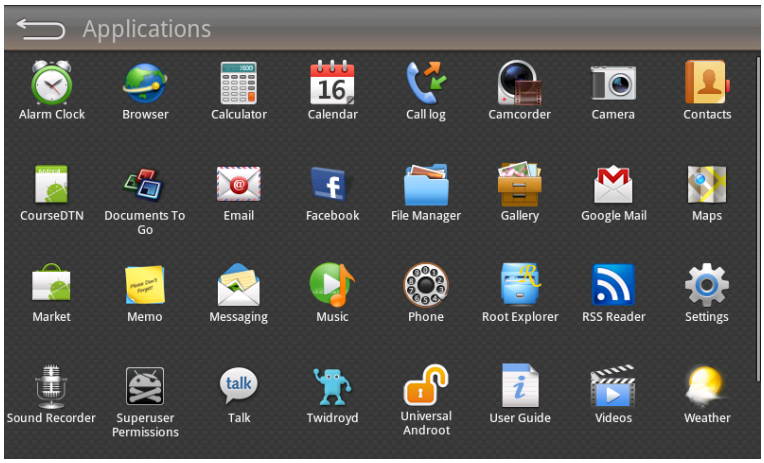

FIGURE 5 – Icone de l'application

Une fois que l'application est lancée, le menu du jeu s'affiche.

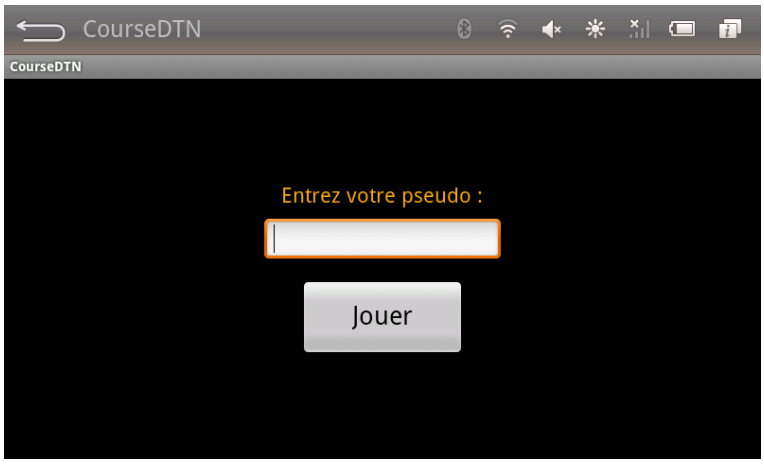

FIGURE 6 – Menu du jeu

Vous pouvez entrer votre pseudonyme dans le champ précédé du label « Entrez votre pseudo : » pour vous identifier par un nom.

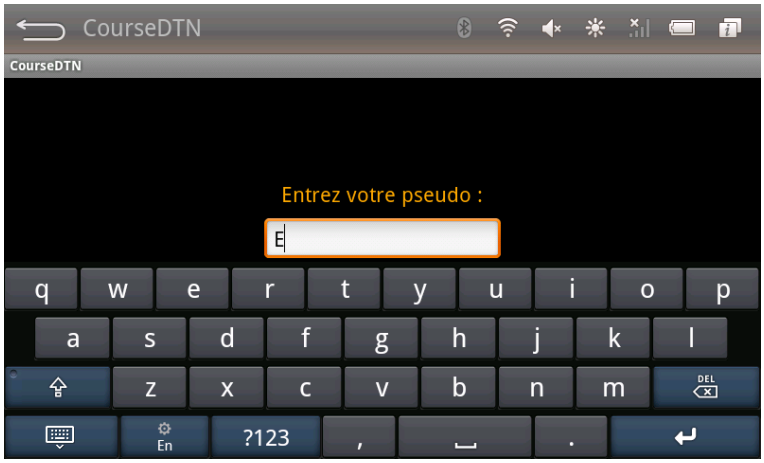

FIGURE 7 – Entrée du pseudonyme

Le bouton « Jouer » lance la partie. Si le bouton jouer est activé sans qu'un pseudonyme ne soit entré, un message d'alerte s'affiche vous indiquant d'entrer un pseudonyme.

FIGURE 8 – Message d'alerte

L'interface du jeu est défini par deux circuits à gauche et à droite identifiant respectivement le joueur local (vous) assimilé à la voiture blanche et le joueur virtuel ou l'adversaire distant assimilé à la voiture rouge. Sur chaque circuit, au plus haut, nous avons de la gauche à la droite, le pseudo et le score. Donc dans le circuit de gauche on a le pseudo local et le score local, et dans le droit, le pseudo adverse et le score adverse.

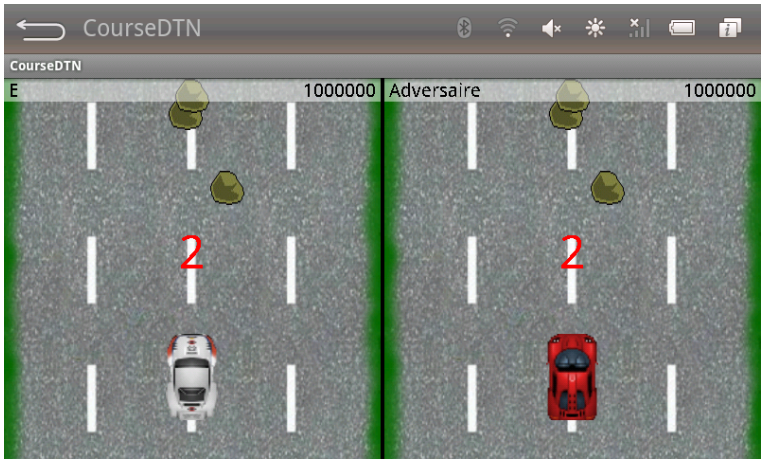

FIGURE 9 – Interface du jeu

Le but du jeu est d'esquiver les obstacles en forme de rochers en effectuant des mouvements d'inclinaison de la tablette afin bouger la position de la voiture. Votre score dépend de votre habilité à esquiver les obstacles.

Sur le circuit gauche, le bouton Wifi identifié par son logo permet d'activer et de désactiver la couche réseau. La couleur bleue indique que le réseau est actif et la couleur rouge qu'il est inactif. Lorsque le réseau est actif un joueur utilisant la même application peut à tout moment se connecter et devenir votre adversaire.

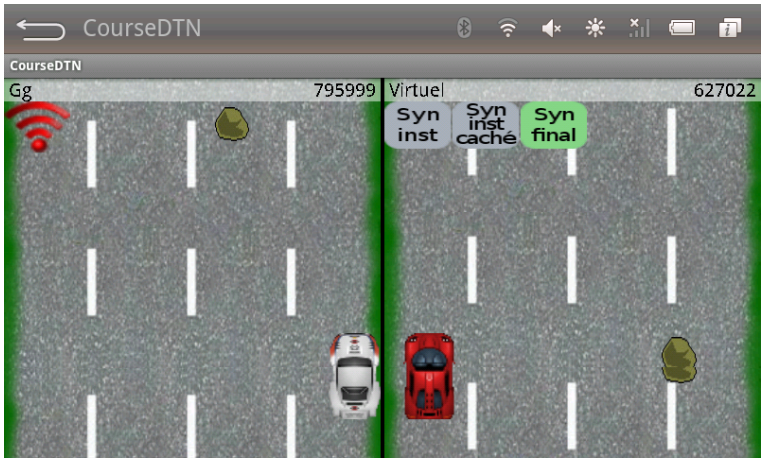

FIGURE 10 – Bouton wifi désactivé

Trois boutons situés sur le circuit adverse (droite) permettent de choisir le mode de synchronisation à effectuer après une connexion. Vous avez donc le choix entre une synchronisation instantanée, instantanée cachée, ou transparente.

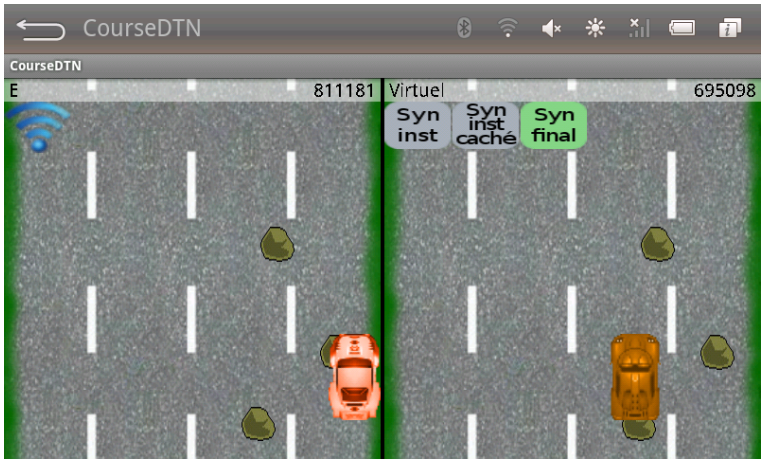

FIGURE 11 – Les boutons de synchronisation

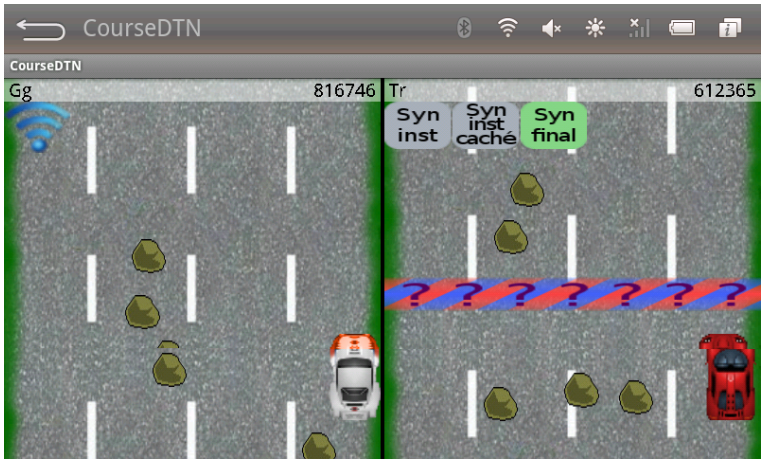

FIGURE 12 – Synchronisation finale avec l'obstacle permettant de gérer le score

Il est à noter que nous observons aussi aléatoirement un brouillard qui cache le circuit pendant un court instant.

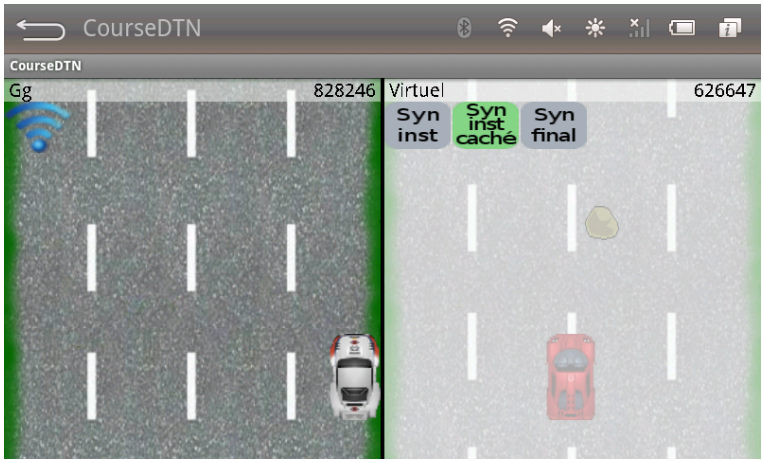

FIGURE 13 – Synchronisation instantanée cachée

En touchant la voiture adverse (en rouge), vous pouvez afficher ou masquer le pseudonyme de l'adversaire. Grâce au pseudonyme vous pouvez savoir si vous jouer contre un joueur virtuel dont le pseudo est « Virtuel » ou contre un joueur distant.

Pour arrêter de jouer, appuyez sur le bouton retour (ou back) de la tablette. Le menu du jeu s'affiche. Une pression supplémentaire sur le bouton retour permet de quitter l'application.

Au cas ou la connexion wifi ne fonctionnerait pas, il faudra juste reconfigurer celui-ci dans le menu « Setting » de la tablette, tout en définissant le réseau choisi pour pouvoir se connecter avec un tiers-utilisateur.

IX Conclusion

Ce projet de course de voiture en DTN nous a permis de mettre en oeuvre une application transparente aux connexions et déconnexions. Tous les objecifs fixés dans le cahier des charges ont été atteints. Par ailleurs l'un des problèmes majeurs que nous avons rencontrés a été la gestion du wifi ad-hoc qui n'est pas pris en compte par l'interface utilisateur de la version d'Android installée sur nos tablettes. Cependant, une autre difficulté mais surmontée a été de concevoir un algorithme de synchronisation qui permet d'apporter une connexion transparente au jeu.

D'autre part, ce projet nous a permis de découvrir le développement d'applications Android et d'appréhender les contraintes qui peuvent se poser par rapport au développement courant. De plus, grâce au projet, nous avons découvert le concept de DTN qui a rendu possible la mise en oeuvre d'une application réseau tolérant au délai.

Références

[1] Android developer. http://developer.android.com/ Lien consulté le 24/03/2011.

[2] Article contenant le binaire du wpa_supplicant modifié. http://www.nkl4.fr/2010/12/07/controler-un-ar-drone-sous-android/ Lien consulté le 24/03/2011.

[3] Get android source. http://source.android.com/source/download.html Lien consulté le 24/03/2011.

[4] Site de l'application barnacle wifi tether développer par szym. ttp://szym.net/barnacle/ Lien consulté le 24/03/2011.

[5] Damine Guignard, Julien Chable, and Emmanuel Robles. *Programmation Android, de la conception au déploiement avec le SDK Google Android 2*. Eyrolles, 2e edition, 2010.